

LAPORAN PRAKTIKUM DATA STRUCTURE

HEAPS

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033



Oleh :

Muhammad Alif Mujaddid

240605110082

Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. PENDAHULUAN

1. Listing program java untuk Heap Berikut ini listing program untuk Heap. Tulis dan pelajari listing ini

<pre>public class Node { private int data; public Node(int key) { data = key; } public int getKey() { return data; } public void setKey(int id) { data = id; } }</pre>	
<pre>public class Heap { private Node[] heapArray; private int maxSize; private int currentSize; public Heap(int size) { maxSize = size; currentSize = 0; heapArray = new Node[size]; } public boolean isEmpty() { return currentSize == 0; } public boolean insert(int key) { if (currentSize == maxSize) { return false; } Node newNode = new Node(key); heapArray[currentSize] = newNode; trickleUp(currentSize++); return true; } public void trickleUp(int index) { int parent = (index - 1) / 2; Node bottom = heapArray[index]; while (index > 0 && heapArray[parent].getKey() < bottom.getKey()) { heapArray[index] = heapArray[parent]; index = parent; parent = (parent - 1) / 2; } heapArray[index] = bottom; } }</pre>	

<pre> public Node remove() { Node root = heapArray[0]; heapArray[0] = heapArray[--currentSize]; trickleDown(0); return root; } </pre>	
<pre> public void trickleDown(int index) { int largerChild; Node top = heapArray[index]; while (index < currentSize / 2) { int leftChild = 2 * index + 1; int rightChild = leftChild + 1; if (rightChild < currentSize && heapArray[leftChild].getKey() < heapArray[rightChild].getKey()) { largerChild = rightChild; } else { largerChild = leftChild; } if (top.getKey() >= heapArray[largerChild].getKey()) { break; } heapArray[index] = heapArray[largerChild]; index = largerChild; } heapArray[index] = top; } </pre>	
<pre> public void displayHeap() { System.out.println("Heap Array: "); for (int i = 0; i < currentSize; i++) { if (heapArray[i] != null) { System.out.print(heapArray[i].getKey() + " "); } else { System.out.println("--"); } } System.out.println(""); int nBlanks = 32; int itemsPerRow = 1; int column = 0; int j = 0; String dots = "....."; System.out.println(dots + dots); while (currentSize > 0) { if (column == 0) { for (int k = 0; k < nBlanks; k++) { System.out.print(' '); } } System.out.print(heapArray[j].getKey()); if (++j == currentSize) { break; } } } </pre>	
<pre> if (++column == itemsPerRow) { nBlanks /= 2; itemsPerRow *= 2; column = 0; System.out.println(); } else { for (int k = 0; k < nBlanks * 2 - 2; k++) { System.out.print(' '); } } System.out.println("\n" + dots + dots); } public void displayArray() { for (int j = 0; j < maxSize; j++) { System.out.print(heapArray[j].getKey() + " "); } System.out.println(""); } } //akhir class Heap </pre>	

Lengkapi listing tersebut dengan sebuah class HeapApp berisi method main. deklarasikan sebuah heap dengan ukuran 35, lakukan penambahan 12 item, tampilkan heap tersebut. Jalankan program, bagaimana output program yang telah anda lengkapi?

```

130 class HeapApp {
    Run | Debug
131     public static void main(String[] args) {
132         Heap heap = new Heap(size:35);
133         heap.insert(key:50);
134         heap.insert(key:30);
135         heap.insert(key:20);
136         heap.insert(key:15);
137         heap.insert(key:10);
138         heap.insert(key:8);
139         heap.insert(key:16);
140         heap.insert(key:60);
141         heap.insert(key:2);
142         heap.insert(key:1);
143         heap.insert(key:100);
144         heap.insert(key:70);
145
146         System.out.println(x:"== Heap setelah 12 insert ==");
147         heap.displayHeap();
148     }
149 }

```

```

=== Heap setelah 12 insert ===
Heap Array:
100 60 70 30 50 20 16 15 2 1 10 8
.....
                        100
                    60      70
                30      50      20      16
            15      2      1      10      8
.....

```

Dari output program tersebut, tampak bahwa heap yang diimplementasikan adalah Heap **MAX** / ~~MIN~~ (coret salah satu) karena elemen terbesar (100) selalu berada di root dan setiap parent memiliki nilai lebih tinggi dari child-nya.

2. Heap Array Perlu diingat bahwa heap adalah sebuah complete binary tree. Dalam bentuk heap (tree) maka akan jelas terlihat parent dan child suatu node. Sedangkan, jika heap tersebut direpresentasikan pada array (sehingga disebut Heap Array), maka visualisasi struktur tree tidak tampak, sehingga bagaimana mengetahui parent dan child suatu node? Pada heap array, tiap node disimpan pada tiap cell array, oleh karena itu, parent dan children suatu node dapat ditelusuri dari posisi indeks node tersebut. Pada program nomor 1, tuliskan kembali tampilan heap dan heap array sesuai data yang telah anda masukkan! Tuliskan indeks tiap item heap array, begitu pula tuliskan indeks item tersebut pada heap. Amati dimana posisi parent dan children suatu node pada heap array! Pada heap Array, beri garis sebagai tanda hubung antara root node dengan left child dan right child dari root. Begitu pula seterusnya, jika right child dari

Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	100	70	60	50	30	20	16	15	2	1	10	8

Index 0 (100)

├ Left Child index 1 → 70
└ Right Child index 2 → 60

Index 1 (70)

├ Left Child index 3 → 50
└ Right Child index 4 → 30

Index 2 (60)

├ Left Child index 5 → 20
└ Right Child index 6 → 16

Index 3 (50)

├ Left Child index 7 → 15
└ Right Child index 8 → 2

Index 4 (30)

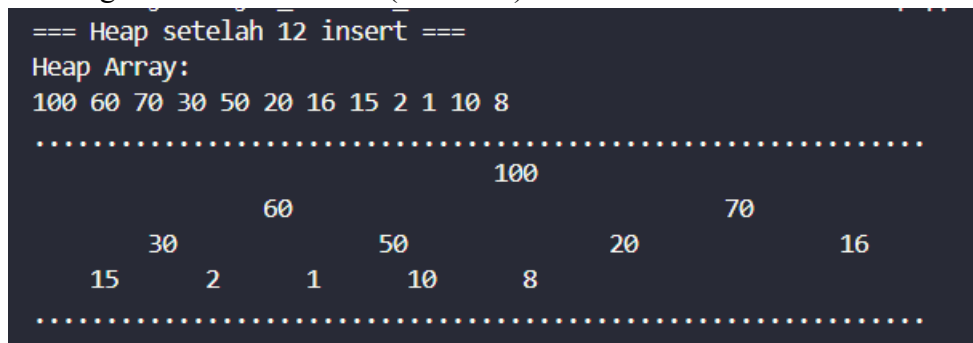
├ Left Child index 9 → 1
└ Right Child index 10 → 10

Index 5 (20)

├ Left Child index 11 → 8
└ Right Child index 12 → (tidak ada, karena size=12)

Index 6 (16)

- └ Left Child index 13 → (tidak ada)
- └ Right Child index 14 → (tidak ada)

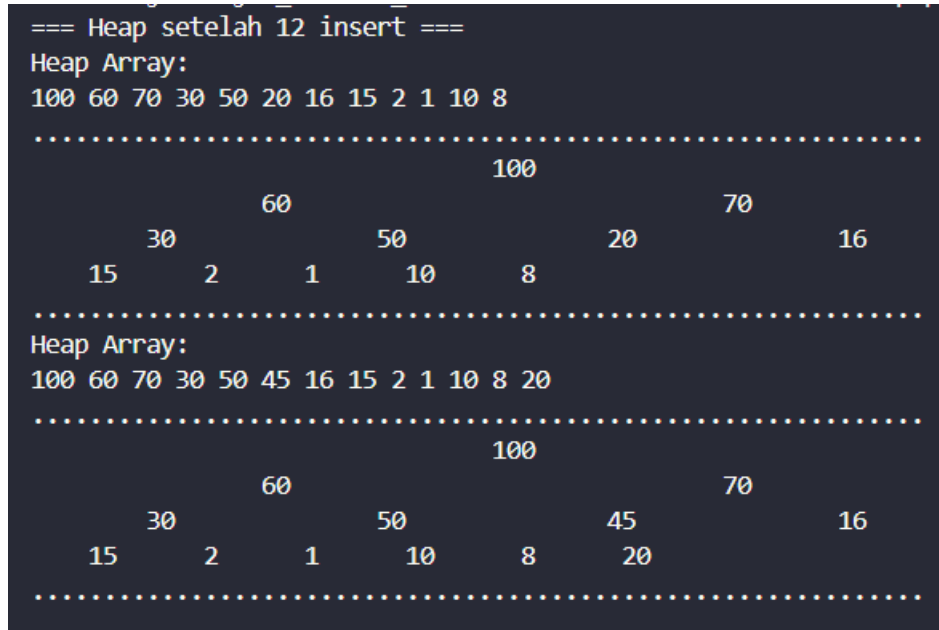


Dari susunan array tersebut diketahui untuk sebuah node pada indeks x, maka:

Indeks Parent	Indeks Children	
x	<i>Left child</i>	$2x + 1$
	<i>Right child</i>	$2x + 2$
$(x - 1) / 2$	x	

3. Menambahkan sebuah item pada Heap Langkah untuk menambahkan item pada heap, yaitu: a. Tambahkan node baru yang merepresentasikan sebuah item dengan key tertentu pada bagian akhir heap, yaitu sebagai last node b. Lakukan trickle up untuk menempatkan node baru tersebut sehingga berada di posisi yang tepat pada struktur heap Trickle, bisa juga disebut dengan istilah bubble atau percolate, yaitu proses untuk memindahkan node baik keatas (trickle up) ataupun kebawah (trickle down) pada path node tersebut secara bertahap dengan cara membandingkan node di tiap tahap apakah node tersebut sudah berapa pada posisi yang sesuai atau belum. Jika posisi tidak sesuai maka dilakukan pertukaran node tersebut dengan node yang dibandingkan hingga menjadi struktur heap yang sesuai. Pada program nomor 1, dari 12 item yang telah ada, lakukan penambahan sebuah item. Tampilkan heap sebelum dan setelah penambahan item. Gambarkan langkahlangkah penambahan item tersebut mulai dari penambahan node baru sebagai last node hingga node tersebut berada pada posisi yang tepat sesuai struktur heap! Beri penjelasan pada tiap langkah tersebut!

```
heap.insert(key:45);
```

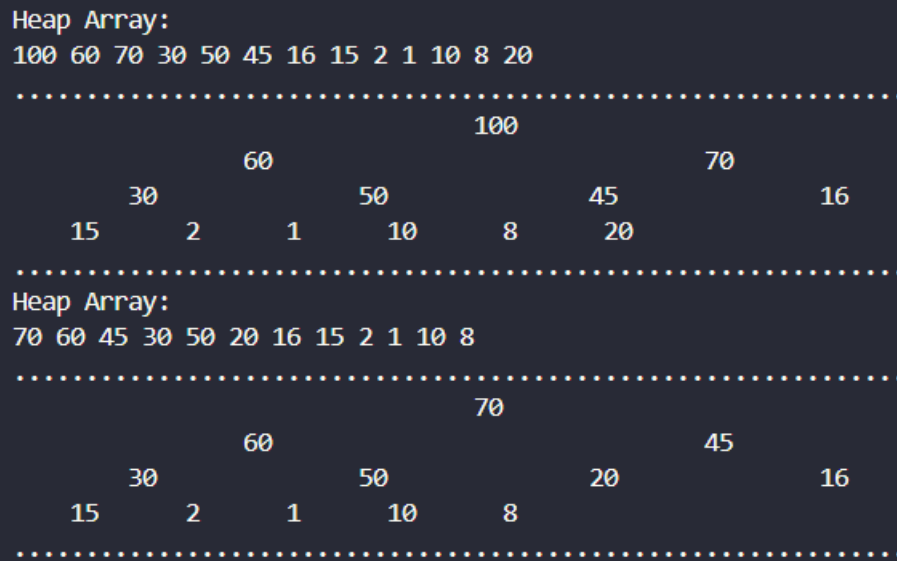


4. Menghapus sebuah item pada Heap Perlu ingat bahwa removal pada heap berarti menghapus node yang memiliki nilai key maksimum (untuk heap MAX) atau menghapus node yang memiliki nilai key minimum (untuk heap MIN). Sehingga node yang dihapus adalah root node. Berikut ini langkah-langkah removal pada heap:

- Hapus root node
- Pindahkan last node pada root
- Lakukan trickle down untuk menempatkan node tersebut pada posisi yang tepat sesuai struktur heap

Pada program nomor 1, panggil method `remove()` pada class `HeapApp`. Tampilkan heap sebelum dan setelah penghapusan. Gambarkan langkah-langkah removal/penghapusan tersebut! Jelaskan tiap tahapnya!

```
heap.displayHeap();
heap.remove();
heap.displayHeap();
```



5. Merubah key (priority) Dengan adanya method `trickleUp()` dan `trickleDown()`, maka algoritma untuk merubah suatu key (representasi dari nilai prioritas pada priority queue) sebuah node dapat mudah diimplementasikan. Secara garis besar, berikut ini langkah untuk merubah key suatu node pada heap: a. Ubah nilai key pada node tertentu (ditunjukkan dengan indeks suatu cell pada heap array yang ingin dirubah) dengan nilai yang baru. b. Jika nilai key yang lama kurang dari nilai yang baru, maka lakukan `trickle up` c. Jika tidak, maka lakukan `trickle down`. Tuliskan sebuah method `change` dengan parameter indeks dan nilai yang baru serta implmentasikan algoritma merubah key pada method tersebut! Panggil method tersebut pada class `HeapApp`, tampilkan heap sebelum perubahan key dan stelah key dirubah

```

public void change(int index, int newValue) {
    if (index < 0 || index ≥ currentSize) {
        return;
    }
    int oldValue = heapArray[index].getKey();
    heapArray[index].setKey(newValue);
    if (oldValue < newValue) {
        trickleUp(index);
    } else {
        trickleDown(index);
    }
}

heap.change(index:2, newValue:110);
heap.displayHeap();

```

70 60 45 30 50 20 16 15 2 1 10 8

.....

70

60

45

30

50

20

16

15

2

1

10

8

.....

Heap Array:

110 60 70 30 50 20 16 15 2 1 10 8

.....

110

60

70

30

50

20

16

15

2

1

10

8

.....